Circuit Reliability Analysis Using Symbolic Techniques

Natasa Miskov-Zivanov, Diana Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
{nmiskov,dianam}@ece.cmu.edu

Abstract - *Due to the shrinking of feature size and significant reduction in noise margins, nanoscale circuits have become more susceptible to manufacturing defects, noise-related transient faults and interference from radiation. Traditionally, soft errors have been a much greater concern in memories than in logic circuits. However, as technology continues to scale, logic circuits are becoming more susceptible to soft errors than memories. To estimate the susceptibility to errors in combinational logic, we propose the use of Binary Decision Diagrams (BDDs) and Algebraic Decision Diagrams (ADDs) for unified symbolic analysis of circuit reliability. We present a framework that uses BDDs and ADDs and enables analysis of combinational circuits reliability from different aspects: output susceptibility to error, influence of individual gates on individual outputs and overall circuit reliability, and the dependence of circuit reliability on glitch duration, amplitude, and input patterns. This is demonstrated by the set of experimental results, which show that the mean output error susceptibility can vary from less then 0.1%, for large circuits and small glitches, up to about 30% for very small circuits and large enough glitches.*

1. Introduction

For the last few decades, the main factors driving the design of digital systems have been cost, performance, and, more recently, power consumption. However, with technology scaling, reliable operation of digital systems is being severely challenged, thus pointing to the use of fault-tolerance-driven design methodologies, not only for mission critical applications (medical, banking, traffic control, etc.), but also for regular, mass-market applications [1].

To allow for the efficient design of a system that can tolerate faults, a first natural step includes understanding the source of induced errors, but most importantly, their modeling and analysis for the purpose of guiding the design process.

A *fault* manifests itself as an incorrect state in the hardware or software that is part of the system. Such faults can result from physical defects, design flaws, or operator errors. According to their source or duration, faults can be divided into permanent, transient and intermittent faults.

- *Permanent* faults occur and remain stable until a repair is undertaken (e.g., stuck-at-zero, stuck-at-one).
- *Transient* (external, soft, or SEU-Single Event Upset) faults occur for a short period of time and then disappear (a bit flip due to a transient physical phenomena, e.g., cosmic ray, alpha particle). These faults can cause an error in the system by changing the internal state, even though they last only for a short time.
- *Intermittent* faults, after they first occur, usually exhibit a relatively high occurrence rate and, eventually, tend to become permanent [2].

Manifestation of a fault is called an *error*, and the system-level effect of an error is known as a *failure*. The principle of fault-tolerance is to automatically surmount the effects of faults by use of redundant components. Consequently, a fault-tolerant system is one, which is capable of continued operation with little or no performance degradation and without corruption of data, in the presence of failure due to either internal or external causes. However, not all faults lead to errors and not all errors lead to failures.

In this work, we address the first issue mentioned above – that is, estimating the likelihood that a transient (physical) fault will lead to an error. Our main goal is to allow for *symbolic modeling and efficient estimation* of the soft error susceptibility of a combinational logic circuit. This can be further used to reduce the cost of applying various techniques for error detection and correction.

1.1. Transient faults in current semiconductor technology

The shrinking of feature size leads to the increase of the amount of charge usually stored in circuit nodes. This increase, together with the significant reduction in noise margins makes circuits more susceptible to manufacturing defects, noise-related transient faults and interference from radiation. When high-energy neutrons or alpha particles hit the silicon bulk, they create minority carriers, which, if collected by a *p-n* junction, result in a current pulse of very short duration. A current pulse that occurs as the result of the strike is often called a SEU (single-event upset). These events may cause a bit flip in some latch or memory element. Additionally, a SEU may occur in an internal node of combinational logic and propagate to the latch. If latched, it results in a *soft error*.

Traditionally, soft errors have been of greater concern in memories than in logic circuits, because of the small cell size of memories and the nature of memory – a SEU can immediately result in a soft error if it exceeds the critical charge stored in the cell. In contrast to this, three factors prevented logic from becoming more susceptible to soft errors:

1. *Logical masking* – to be latched, a SEU needs to be on the sensitized path from the location where it originates to the latch;
2. *Electrical masking* – a SEU needs to create a pulse that has a duration and amplitude large enough to reach the latches. Due to the electrical properties of the gates the pulse (glitch) is passing through, it can be attenuated and even completely masked before it reaches the latch;
3. *Latching-window masking* – if the pulse reaches the latch and appears at its input “on time” (during this window), depending on its amplitude and duration, it has a great probability of being latched.

However, as technology continues to scale, logic circuits are becoming much more susceptible to soft errors. The trends toward reduced logic depth reduce the attenuation when SEU is propagating through the circuit. Smaller feature sizes and lower voltage levels allow lower energy particles to cause SEUs. Therefore, soft error failure rates in combinational logic are expected to become very important in the future [3] and even exceed soft error rates in memories.

1.2. Paper organization

The rest of this paper is organized as follows. In Section 2 we give an overview of related work. Section 3 describes our assumptions and the notations we use in the rest of the paper. Section 4 presents in more detail the mathematical model that lies behind our framework. In Section 5, we describe our symbolic modeling methodology, while in Section 6 we describe a practical method for determining circuit susceptibility to soft errors. In Section 7, we report experimental results for a set of common benchmarks. Finally, with Section 8 we conclude our work and provide some directions for future work.

2. Related work

2.1. Transient fault analysis and modeling.

Intensive research has been done so far in the area of analysis and modeling transient faults [3-5,7-9]. However, for estimating the likelihood of soft errors as the result of a SEU, most of the previous work has relied on fault injection [1,6,7] and simulation instead of the symbolic modeling of the probability of soft errors. The results presented by Mohanram *et al.* [1] show that soft error susceptibility of internal nodes in a logic circuit can vary by at least one order of magnitude. Based on this fact, the authors have applied concurrent error detection (CED) techniques asymmetrically (targeting mostly the nodes with high soft error susceptibility), which led to reduced cost.

In [7], the authors give a mathematical model for analyzing the propagation of a transient fault through a chain of combinational gates. They verified that their model has 90% average accuracy with respect to HSPICE simulation. However, their work was focused on estimating electrical masking on the sensitized path in the circuit, while logical and latching-window masking were not included.

Work by Zhao *et al.* [8], also stressed the importance of analyzing the effect of internal glitches on the latched outputs of the circuit. For electrical masking, the authors use noise rejection curves and find the probability that noise will propagate through the given node, without being completely attenuated. Each node is analyzed separately, so their analysis does not reflect the influence of the location of the node inside the circuit on the observability of the noise at the latched output. Moreover, for logical masking, the authors use path tracing, which can become very inefficient for larger circuits.

In [9], Zhang *et al.* present a methodology for soft error rate analysis. This work focuses mostly on modeling the probability that a single event transient is generated by a particle hit. Electrical masking for each path is obtained from HSPICE simulation, and logical masking is computed for each input vector and each path separately, by flipping the logic value of each node.

Two more recent works on reliability evaluation have been presented [10,11]. In [10], Dhillon *et al.* present an *independent* computation of the three factors, logical, electrical, and latching-window masking to find the soft-error tolerance of the circuit. Work by Krishnaswamy *et al.* in [11] uses probabilistic transfer matrices and their representation via Algebraic Decision Diagrams. Each gate can be represented as a matrix where the probability of each output value is explicit for each input combination. Parallel compositions of gates are represented with tensor products. However, the work presented in [11] focuses *only* on *logical masking* effect of the circuit for given gate output probabilities, without considering electrical and latching-window masking.

2.2. Analysis of combinational circuits using BDDs and ADDs

In order to estimate the probability of errors in combinational logic, our symbolic tool uses Binary Decision Diagrams (BDDs) and Algebraic Decision Diagrams (ADDs), as part of the CUDD package [12]. BDDs [13,14] provide an efficient and canonical representation for Boolean functions. In [15], a new type of BDD, called a MultiTerminal BDD (MTBDD), was introduced. A MTBDD allows for multiple terminal nodes in the canonical representation. Similar to MTBDDs, ADDs [16] are presented as a class of symbolic models and associated algorithms applicable not only to arithmetic, but also to many algebraic structures. For example, these decision diagrams were applied to symbolic timing analysis in [17]. In that work, the authors present RESTA, a robust and extendable timing analysis tool that addresses three main goals: considers both internally and externally specified input constraints, handles a wide range of circuit structures and have a robust underlying framework. This application has shown ADDs to be practical and efficient, while providing quite accurate results.

2.3. Paper contribution

There are some important differences between our model and those in [8-11]. In comparison to [8-10], where latching-window, electrical and logical masking are analyzed separately and assumed independent, our approach provides a unified treatment of these three factors, while including their *joint* dependency on input patterns and circuit topology. In most of the previous work, information about electrical masking is obtained by simulation [9], while information about logical masking is obtained by path tracing [8-10]. In our work, by using BDDs and ADDs, this information is instead implicitly included inside the decision diagrams, and therefore allows for efficient concurrent computation of output error susceptibility due to hits on various internal nodes. In the case of reconvergent glitches (that is, glitches arriving at the same gate or latched output from the same source on two or more different sensitized logical paths), the problem of merging the glitches needs to be addressed. In [8], a similar problem for several different noise sources is solved by shifting the noise rejection curve. The authors in [9] approximate the case of reconvergent glitches with the worst case, and claim that in most cases this does not affect the accuracy significantly. Our approach to this problem is different from these two and is explained in more detail in Section 5. Finally, while [11] provides a symbolic method for circuit reliability, it does not include the additional joint impact of electrical and latching window masking and presents logical masking only.

3. Assumptions and notations

We show in Figure 1 an example of a target circuit we are analyzing, including the combinational logic, as well as its input and output latches. We estimate the probability that a pulse or glitch, occurring due to some transient physical phenomenon at an internal gate G of the circuit, will result in an error at output F. In our framework, we capture all *gate-output* combinations, i.e., we determine the probability of a soft error at any output due to a fault originating at any internal gate.

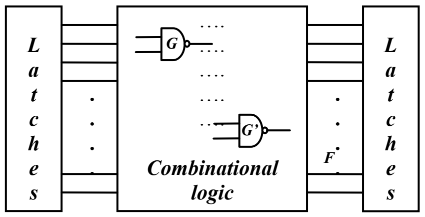

Figure 1. A target combinational circuit with input and output latches.

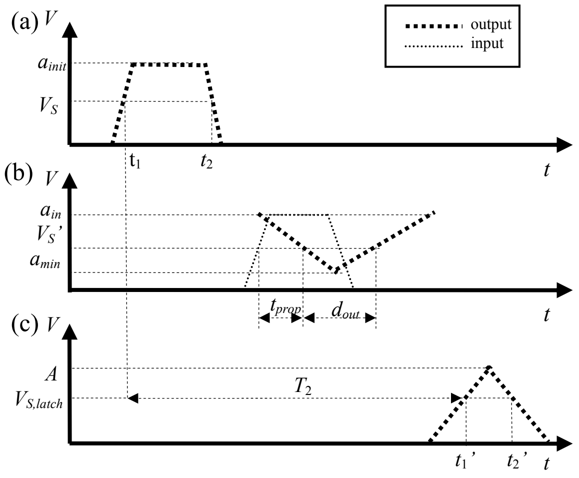

Figure 2. A glitch at the (a) output of initial gate G, (b) input and output of gate G' on sensitized path, and (c) circuit output F.

Figure 2 shows the propagation of the glitch, that is, the shape at the output of gate G where it occurs (Figure 2a), at the input and output of a gate G' on the sensitized path between gate G and latched output F (Figure 2b), and at the output F (Figure 2c).

At the output of gate G, the glitch has initial duration d_{init}, and initial amplitude a_{init}. The duration at the output of the gate is always measured at switching threshold voltage (V_S) [18] of downstream gate, therefore, according to Figure 2:

$$d_{init} = t_2 - t_1 \tag{1}$$

The propagation of a glitch through an internal gate G' (V_s') is shown in Figure 2b. At the input of gate G', the glitch has amplitude a_{in} and duration d_{in}, and the output amplitude a_{out} and duration d_{out}. Durations d_{in} and d_{out} are in this case measured at the switching threshold voltage of gate G' [18]. However, for all output neighbors of gate G', d_{out} will be recomputed according to their switching thresholds. Propagation delay of gate G' is t_{prop}. To find out if the glitch propagates through gate G', and to compute the new amplitude and duration, we use the methodology from [7], as explained in Section 4. Finally, at the latched output F, the glitch has amplitude A and duration D. Switching threshold voltage of the latch, at which D is measured, is $V_{S,latch}$. Since there is a delay from gate G to output F (T_2), the time when the glitch becomes larger than $V_{S,latch}$ is t_1', and when it becomes lower than $V_{S,latch}$ is t_2':

$$T_2 = t_1' - t_1 \tag{2}$$

$$D = t_2' - t_1' \tag{3}$$

The duration D, as well as the amplitude A, can have different values at output F, depending on the various sensitized paths, from G to F. The set of different values of duration D for various sensitized paths is denoted by $\{D_k\}$. The delay T_2 depends on the sensitized path (i.e., on the gate delays on that path) from gate G to output F, while the delay from input latches to gate G (T_1) depends on the path from inputs to gate G. However, in our model, when computing latching window masking, we assume the worst case in which the latching window probability is maximized, as it will be seen next.

Since we are interested in the propagation of a glitch in the time interval between two rising edges of the clock signal, we can take $[0, T_{clk}]$ as the interval of observation. For a signal to be latched, it needs to be stable during the setup time t_{setup} before the rising edge of the clock, and hold time t_{hold} after the rising edge of the clock. In other words, it needs to be stable inside interval $[T_{clk}-t_{setup}, T_{clk}+t_{hold}]$.

4. Mathematical description of the model

This section describes the conditions that are needed for a transient glitch at the output of an internal gate to be propagated to the output and latched, such that a soft error is registered. We detail the interdependency between conditions for logical, electrical, and window masking, and describe their joint model.

4.1. Necessary conditions

To this end, we define the following events:

$\mathcal{E}$ – a glitch originating at gate G is latched at output F;

$\mathcal{A}$ – the amplitude of a glitch at the output is larger than the switching threshold of the latch (in case when correct output value is "0") or smaller than the switching threshold (in case when the correct output value is "1");

$\mathcal{D}$ – the duration of a glitch at the output is larger than the sum of setup and hold time of the latch;

$\mathcal{T}$ – the glitch appears at the output on time to be latched (i.e., it satisfies the setup time and hold time conditions when the rising edge of the clock occurs).

It is clear that for event $\mathcal{E}$ to happen, the other three events need to occur:

$$\mathcal{E} = \mathcal{A} \cap \mathcal{D} \cap \mathcal{T} \tag{4}$$

In this model, logical and electrical masking are implicitly included in $\mathcal{A}$ and $\mathcal{D}$, while latching window masking is included in $\mathcal{T}$. As mentioned in Section 3, the switching threshold of the latch at output F is $V_{S,latch}$. To satisfy the latching condition, the time at which the glitch reaches $V_{S,latch}$ (t_1') must satisfy:

$$t_1' < T_{clk} - t_{setup} \tag{5}$$

In addition, the time when the glitch becomes less than $V_{S,latch}$ (t_2') must satisfy:

$$t_2' > T_{clk} + t_{hold} \quad (6)$$

with duration D of the glitch at output F given by equation (3). Thus, we can write the condition which allows a glitch occurring at gate G to be latched, as:

$$t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2) \quad (7)$$

More formally, one can express the three events as follows:

$\mathcal{A}$: $A > V_{S,latch}$ (when correct output value is "0") or
$A < V_{S,latch}$ (when correct output value is "1")

$\mathcal{D}$: $D > t_{setup} + t_{hold}$

$\mathcal{T}$: $t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2)$

Therefore, the probability of event $\mathcal{E}$ can be written as:

$$P(\mathcal{E}) = P(\mathcal{A} \cap \mathcal{D} \cap \mathcal{T}) = P(\mathcal{T} | \mathcal{A} \cap \mathcal{D}) \cdot P(\mathcal{A} \cap \mathcal{D}) \quad (8)$$

As seen in Figure 2c, $\mathcal{D}$ is satisfied only if $\mathcal{A}$ is satisfied, that is, only if the amplitude of the glitch is larger than the switching threshold, the duration can be different from zero, then :

$$\mathcal{D} \subset \mathcal{A} \quad (9)$$

and thus:

$$\mathcal{A} \cap \mathcal{D} = \mathcal{D} \quad (10)$$

which implies:

$$\begin{aligned} &P(\mathcal{E}) = P(\mathcal{T} | \mathcal{D}) \cdot P(\mathcal{D}) = \\ &P(t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2) \cap D > t_{setup} + t_{hold}) = \\ &P(t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2) \cap (\bigcup_k D = D_k)) = \\ &\sum_k (P(t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2) \mid D = D_k) \cdot P(D = D_k)) \end{aligned} \quad (11)$$

where $\{D_k\}$ is the set of possible glitch durations, along various sensitized paths.

We assume that t_1 is uniformly distributed in the interval $(T_1, T_1 + T_{clk} - d_{init})$. Thus, in the worst case when, for a given glitch duration D_k, the interval $(T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2)$ lies inside it, the probability of event $\mathcal{T}$ at the output is:

$$\begin{aligned} &P(t_1 \in (T_{clk} + t_{hold} - T_2 - D, T_{clk} - t_{setup} - T_2) \mid D = D_k) = \\ &\frac{D_k - (t_{setup} + t_{hold})}{T_{clk} - d_{init}} \end{aligned} \quad (12)$$

4.2. The attenuation model

From previous equations we can see that, to determine the probability of event $\mathcal{E}$, it is necessary to find out what are the possible values for duration, $\{D_k\}$, and determine the probabilities associated with those values. Another issue is finding the correct values for amplitude at the output. To find these values, we use the method proposed in [7]. Figure 2 shows how the glitch propagates from the output of gate G to the output of a gate G', which is assumed to be part of the sensitized path from G to a generic output F.

As claimed in [7], when the glitch propagates to the input of gate G', depending on the relation between the duration d_{in} of the glitch and the propagation time of the gate G', t_{prop}, there are three possible options:

- if $d_{in} \le t_{prop}$, then the glitch will not propagate through the gate (it is masked);
- if $t_{prop} < d_{in} \le 2t_{prop}$, then the glitch will propagate, but the amplitude and the duration will be smaller at the output of a gate (it is attenuated);
- if $2t_{prop} < d_{in}$, then the glitch will not be attenuated and it will be propagated as is.

As it can be seen, the amplitude and the duration of the glitch at the output of the gate through which the glitch propagates depend on the input glitch duration, amplitude, and propagation delay of gate G'. However, if the output glitch amplitude a_{out} is not larger than the threshold for the downstream gate, then it can be assumed that the glitch does not propagate at all. As in [7], we assume the following: when the output voltage has a "1" logic value (V_{dd}), and a glitch affects the input, the output minimum value is:

$$a_{out} = V_{\min} = \begin{cases} V_{dd} & a_{in}/V_S' < VT_1 \\ \dfrac{-V_{dd}}{VT_2 - VT_1} \cdot \dfrac{a_{in}}{V_S'} + \dfrac{V_{dd} \cdot VT_2}{VT_2 - VT_1} & VT_1 < a_{in}/V_S' < VT_2 \\ 0 & a_{in}/V_S' > VT_2 \end{cases} \quad (13)$$

Similarly, when the output voltage has a "0" logic value (0) and a glitch affects the input, the output maximum value is:

$$a_{out} = V_{\max} = \begin{cases} 0 & a_{in}/V_S' < VT_1 \\ \dfrac{V_{dd}}{VT_2 - VT_1} \cdot \dfrac{a_{in}}{V_S'} - \dfrac{V_{dd} \cdot VT_1}{VT_2 - VT_1} & VT_1 < a_{in}/V_S' < VT_2 \\ V_{dd} & a_{in}/V_S' > VT_2 \end{cases} \quad (14)$$

where VT_1 and VT_2 are the thresholds that divide the interval in which a_{in}/V_S' can take values, into three parts. These thresholds are functions of the glitch duration, normalized with respect to the gate propagation delay t_{prop}. The curve obtained from simulation can be approximated by the following equations:

$$VT_1 = c_1 \frac{d_{in}}{t_{prop}} + c_2 \left(\frac{d_{in}}{t_{prop}} \right)^2 + c_3 \left(\frac{d_{in}}{t_{prop}} \right)^3 + b \quad (15)$$

$$VT_2 = c_1' \frac{d_{in}}{t_{prop}} + c_2' \left(\frac{d_{in}}{t_{prop}} \right)^2 + c_3' \left(\frac{d_{in}}{t_{prop}} \right)^3 + b' \quad (16)$$

Coefficients c_i, c_i' and b, b' are determined as in [7] by fitting the equations with results obtained by HSPICE simulations of a simple inverter chain. This attenuation model has been shown to have the average accuracy of 90% when compared to HSPICE. The regions where the model gives less accurate results are around VT_1 and VT_2, but these regions are not of interest, since the glitch certainly does not propagate, or it propagates without attenuation.

5. The symbolic modeling framework

To find the probability of event $\mathcal{E}$ (as described in Section 4.1.), we need to find the possible values for the duration and amplitude of a glitch at the generic output F. To determine the probability of having a glitch of duration D_k at that output, we use BDDs and ADDs. Our algorithm is described in the following.

5.1. ADD creation

ADDs are created starting with the first node in topological order. Duration and amplitude ADD are the same, except for the values stored in the terminal nodes. Terminal node "0" represents combinations of inputs that logically mask the glitch, and all the cases when the glitch becomes too short or too attenuated to be propagated, i.e., all cases when glitch is electrically masked. The values on the other terminal nodes will depend on the paths through which the glitch propagates.

```
createAllADDs {
   set technology parameters;
   parse input netlist;
   create gate node list;
   **for each** *gate* in *gate_node_list*
      build neighbors list;
   sort gates topologically;
   **for each** *gate* in *sorted_gate_node_list*) {
      create output BDD;
      find reconvergent paths;
      merge ADDs;
      create sensitization BDDs;
      create duration and amplitude ADDs;
      remove zero ADDs;
      pass all ADDs to output neighbors;
   }
}

findProbabilities {
   set probabilities for inputs;
   **for each** *output* of the circuit
      **for each** *gate*
         compute the probability of error;
}
```

Figure 3. The *main* algorithm.

The initial ADD for each gate is built for the glitch originating at that gate. It consists of only one terminal node for all possible input patterns – initial duration or amplitude value. Those ADDs are passed to all fanout gates, which use them for creating new ADDs based on their own attenuation model.

Let us now assume that gates *G* and *G'* are internal gates on the sensitized path through which the glitch propagates to the output *F*. To create new ADDs for gate *G'*, we use propagated ADDs from gate *G* (which will propagate the initial glitch amplitude and duration ADDs, but also ADDs that it has built with respect to ADDs passed from its fanin gates), and *sensitization* BDDs. Since the glitch propagates only if it is on a sensitized path, we need to create sensitization BDDs to find out for which input patterns the path between gates *G* and *G'* is sensitized. Thus, to build new ADDs for gate *G'*, we use an ADD received from its input neighbor *G*, and a sensitization BDD, that represents the function $f=\partial G'/\partial G$. Only for the cases that end up in the terminal node "1" in the sensitization BDD, and a node different then "0" in the ADDs, we calculate new values for duration and amplitude. All other cases represent either logically or electrically masked values. Starting with the first node in the topologically sorted list, we create ADDs and BDDs at each node, but they are destroyed as soon as they are not needed. Moreover, some of the current ADDs become "0" due to masking effects, so those ADDs are also removed. When the final node in the circuit is reached, only the ADDs for output *F* are needed.

Each of these ADDs represents a pair *gate-output*, where *gate* is the one where glitch appears and *output* is the one for which we determine the probability of error susceptibility. The terminal nodes for these ADDs represent the final duration or amplitude of a glitch at the output. In addition to them, we also keep track of a list of delays that are computed in parallel with creating ADDs. The delays are used for cases when glitches from reconvergent paths are merged.

To show how our method works, Figure 5 presents ADDs that are built on paths 1→5 and 2→3→5 of the *ISCAS'85* benchmark *C17* (Figure 4). Figure 5a shows sensitization BDDs for paths 1→5 and 2→3→5, while Figures 5b and 5c represent initial and propagated duration ADDs for glitches originating at gate **2** (2 steps) and gates **1** and **3** (one step for each). As it can be seen from Figure 3, the algorithm for creating ADDs is linear in the number of gates and number of inputs, while the algorithm for computing probabilities is linear in number of gates and number of outputs.

In the next section, we explain how glitches arriving on reconvergent paths are merged.

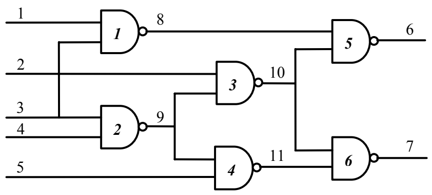

Figure 4. An example circuit (*C17*).

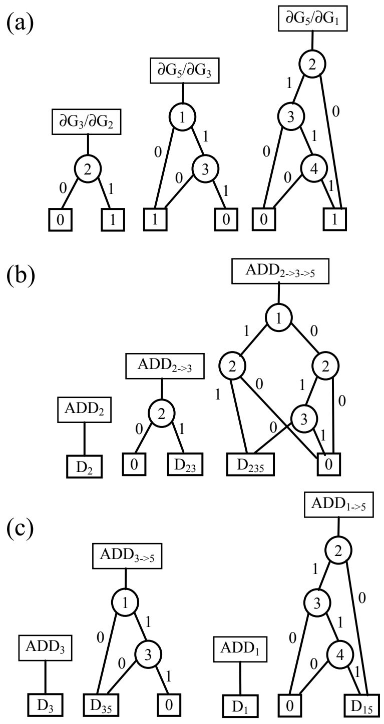

Figure 5. (a) Sensitization BDDs for paths 1→5 and 2→3→5, (b) duration ADDs for the propagation of glitch originating at gate **2**, and (c) duration ADDs for glitches originating at gates **1** and **3**.

5.2. Reconvergent glitches

We define a function *mergeADDs,* which is used to find and merge all ADDs that represent reconvergent glitches. For example, in the case of benchmark *C17*, we can see that the output of gate **2** goes to gates **3** and **4**, and that the outputs of these gates (**3** and **4**) are inputs to gate **6**. Thus, a glitch occurring at the output of the gate **2** can propagate through two paths (through gates **3** and **4**) to gate **6**. In this case, depending on the values on the circuit inputs, different superposition of the two glitches arriving to the inputs of the gate **6** can occur. Therefore, when building ADDs for duration and amplitude, we need to know whether such situations occur, in order to compute the correct values. The pseudocode for the function that merges reconvergent glitches is given in Figure 6.

```
mergeADDs() {
   for each gate {
      create its list of reconvergent paths;
      create quasi sensitization BDD;
      find mutual masking;
      mask reconvergent ADDs;
      sort list according to delays;
      for (i=1 to list size-1) {
         for (j=i+1 to list size) {
            case {
               (value[i] is controlling and value[j] is controlling):
                    merge both controlling;
               (value[i] is non- controlling and value[j] is non-controlling):
                    merge both non-controlling;
               (value[i] is non-controlling and value[j] is controlling):
                    merge non-controlling controlling;
               (value[i] is controlling and value[j] is non-controlling)
                    merge controlling non-controlling;
            }
            update list;
         }
      }
   }
}
```

Figure 6. The algorithm for *mergeADDs* function.

In more detail, from the list of all reconvergent paths arriving to that gate, we analyze separately groups of paths that originate at the same gate. For the paths with the same start gate, and their corresponding ADDs (i.e., glitches), we build a *quasi sensitization* BDD, that is, a BDD where the *zero* node represents all the cases where at least one of other inputs (that do not carry a glitch) is controlling, and one where neither one of them is controlling. This BDD can reduce ADD size, so that we analyze only cases where glitches affect output of the gate. Next, we find the cases where inputs that carry glitches mask each other. Not all reconvergent glitches occur for all input patterns, and there are situations where one of the glitches appears at the gate input, but at least one of the others is logically masked. Moreover, if this input is set to a controlling value, then the existing glitch will also be masked. Thus, we need to mask all these cases in the ADDs for the reconvergent paths. When masking is done, the only input combinations that lead to non-zero terminal nodes in ADDs are those that allow glitches to affect the output of a gate. Each ADD has a delay associated with it, so the list of reconvergent paths is sorted according to their delays. From the sorted list, we take pairs of ADDs and merge them, as shown in Figure 6. When merging ADDs, four possible situations can occur, as shown in Figure 7.

Two inputs (that carry reconvergent glitches) that are to be merged can be: both controlling, both non-controlling, or first can be controlling, second non-controlling and vice versa. It is easy to conclude from Figure 7 that, in some cases, the resulting glitch is the same as one of the original two – we just need to keep that one and mask those cases in ADDs of the other one. The same should be done when a new glitch starts at the same time as one of the original two, except that in this case the corresponding value in duration ADD is changed. There are also situations when glitches mask each other or when one glitch is changed (attenuated), the other one is removed, and a new one appears. Since the new glitch has a new delay, we cannot merge it into neither one of the ADDs, but we need to create a new ADD with the new corresponding delay. If the two resulting glitches are close enough to each other, we can assume they are merged into a single long glitch (worst-case approximation). As it can be seen by direct inspection, the algorithm for merging ADDs on reconvergent paths is linear in the number of outputs, number of gates, and number of reconvergent paths in the circuit.

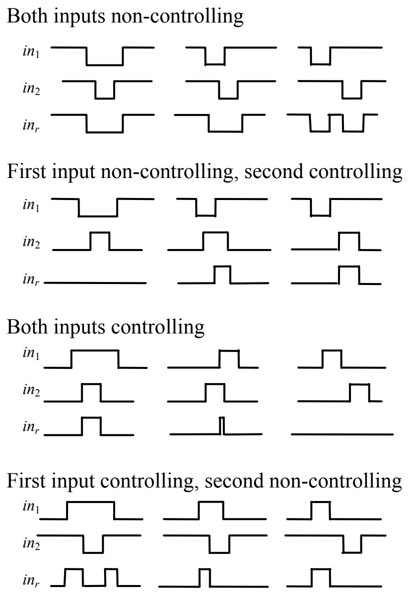
Both inputs non-controlling
in_1
in_2
in_r
First input non-controlling, second controlling
in_1
in_2
in_r
Both inputs controlling
in_1
in_2
in_r
First input controlling, second non-controlling
in_1
in_2
in_r

Figure 7. Possible combinations of reconvergent glitches (in_1 and in_2 are inputs, in_r is the effective glitch)

6. Probability computation

Since different combinations of "0"s and "1"s can occur at the inputs of a given combinational circuit, we set various values for the probability of each input being "1." We use these probabilities to find the error susceptibility for each output of the combinational logic.

When all ADDs for a given circuit are built, the error susceptibility for each output due to an error at the output of any gate in the circuit can be computed. We use equation (11) to compute these probabilities. For a generic output F_j and a gate G_i we build all ADDs representing the duration and amplitude of a glitch starting at the output of gate G_i and propagating to output F_j. Given the probability of "1" for each input, we compute the probability that the glitch duration D at the output is D_k, and the corresponding latching probability for this specific duration value as in equation (12). To analyze error susceptibility of a given combinational logic circuit, we assume a discrete set of test glitches of different initial duration d_{init}, and we use randomly generated input probability distributions. We analyze each circuit from two aspects: reliability of its outputs when faults occur inside the circuit, and influence of individual gates error on outputs. For each output F_j and d_{init}, we find *mean error susceptibility* as the probability of output F_j failing due to errors in the internal gates as:

mean error susceptibility (F_j) =
($\sum_{\text{all } G_i}$ probability of F_j failing if G_i fails)/
(#gates · #prob. distributions) (17)

For each gate G_i and d_{init}, we find minimum, maximum, average and median error susceptibility over all outputs F_j that are affected by a glitch occurring at the output of gate G_i. *Mean error impact* for gate G_i is computed as:

mean error impact (G_i) =
($\sum_{\text{all } F_j}$ probability of F_j failing if G_i fails)/
(#outputs · #prob. distributions) (18)

For each input probability distribution used, we also find the number of gates that do not affect any of the outputs.

7. Experimental results

In this section, we compare the results of our symbolic framework for eight combinational circuits, given different glitch durations and different sets of input probabilities. The technology used is 70nm, Berkeley Predictive Technology Model [19,20]. The clock cycle period (T_{clk}) used is 250ps, and setup (t_{setup}) and hold (t_{hold}) times for the latches are assumed to be 15ps each. V_{dd} is assumed to be 1V, and for simplicity, all switching threshold voltages, gate threshold (V_S', and V_S), and latch threshold $V_{S,latch}$ are assumed to be $V_{dd}/2$. The delay of an inverter in the given technology is determined by simulating a ring oscillator in HSPICE and found to be 10.2ps. The delays for other gates are found by using logical and electrical effort methodology [21]. The benchmark circuits are chosen from *ISCAS'85* and *mcnc'91* suites. Our symbolic modeling framework is implemented in C++, and run on a 3GHz Pentium 4 workstation running Linux.

We have compared glitch durations and delays (obtained using our symbolic framework) at the outputs of circuits *C17* and *circ*, with results from HSPICE simulations for several initial glitch durations. Existing terminal nodes in output duration ADDs in our framework, as expected, match exactly the results of HSPICE simulations in case of logical masking. The electrical masking results also show a good approximation. In most cases, our framework captures reconvergent and merged glitches as well as HSPICE, the only discrepancies coming from the approximate delay and attenuation models used by the symbolic framework.

The results for one small benchmark *5xp1* (116 gates) and one larger benchmark, *C1908* (384 gates) are presented in Figure 8. We divide interval [0,1] of possible error impact into ten subintervals. For each benchmark, each error impact interval, and various input probability distributions, we show the number of gates that have minimum, maximum, mean or median error impact in those intervals. We present this dependence in case of three different initial glitch durations. For the small glitch that has duration of 50ps, all error impact values are in the range from 0 to 0.4. The gates that influence outputs are just the output gates, and their fanin gates. In case of a larger circuit, there is a significant number of gates that do not have any impact on output error. However, in case of a 125ps long glitch, it might not propagate to the output due to logical masking, or it will not be latched due to latching window masking. Since the glitch is very long even at the output, there is a considerable number of gates that will almost certainly have an impact on output error.

In Table 1 we show the experimental results for several benchmarks of varying complexity. We present minimum, maximum, average and median output error susceptibility for all benchmarks, as well as the associated run time. As it can be seen from the results, the mean error susceptibility decreases with the circuit complexity, due to more probable electrical and logical masking. The results also show that median value is usually closer to the minimum. Therefore, we can conclude that most of the outputs have small error susceptibility, but in the case of large glitches (Figure 8), almost all gates have an impact on output failure.

Table 1. Minimum, maximum, average and median circuit error susceptibility computed as in equation (17), for several benchmarks.

Bench.	no. gates	no. PIs	no. POs	glitch size	mean error susceptibility				run time (s)
					min	max	average	median	
z4ml	45	7	4	small	0.00106	0.01003	0.00468	0.00381	0.10
				medium	0.02659	0.04471	0.07187	0.07403	0.20
				large	0.10836	0.31161	0.20181	0.19364	0.28
add16	80	33	17	small	0.00060	0.00185	0.00067	0.00060	2.61
				medium	0.00574	0.04162	0.02084	0.02053	11.30
				large	0.01900	0.11867	0.08414	0.08725	11.35
5xp1	116	7	10	small	0.00041	0.00643	0.00291	0.00224	0.30
				medium	0.00254	0.04193	0.01960	0.01592	0.30
				large	0.00655	0.10108	0.04765	0.03765	0.35
9symml	145	9	1	small	0.00035	0.00035	0.00035	0.00035	0.60
				medium	0.01199	0.01199	0.01199	0.01199	1.25
				large	0.03394	0.03394	0.03394	0.03394	1.40
C499	352	41	32	small	0.00027	0.00113	0.00057	0.00027	4.76
				medium	0.00152	0.00528	0.00293	0.00174	4.90
				large	0.00652	0.02368	0.01131	0.009215	174.00
C1908	384	33	25	small	0.00025	0.00119	0.00044	0.00025	20
				medium	0.0015	0.01089	0.00375	0.00184	176.00
				large	0.00519	0.03906	0.01483	0.00622	672.00
duke2	469	22	29	small	0.00010	0.00049	0.00021	0.00017	3.00
				medium	0.00041	0.00167	0.00087	0.00085	3.50
				large	0.00118	0.00462	0.00247	0.00241	4.43
alu4	560	14	8	small	0.00007	0.00018	0.00009	0.00007	5.00
				medium	0.00027	0.01656	0.00469	0.00330	20.10
				large	0.00246	0.04267	0.01593	0.01201	32.30

8. Conclusion and future work

In this paper, we present a symbolic modeling methodology and associated framework for efficient estimation of the soft error susceptibility of a combinational logic circuit. We have demonstrated the efficiency of our framework by applying it on a subset of *ISCAS'85* and *mcnc'91* benchmarks of various complexities. The framework allows for the analysis of reliability of combinational circuits from various aspects: output susceptibility to error, influence of individual gates on individual outputs and overall circuit reliability, and the dependence of the circuit reliability on glitch duration, amplitude, and input patterns.

An area of future research is the use of more accurate model for latching window masking, by taking into account all possible values of propagation delays on the sensitized paths.

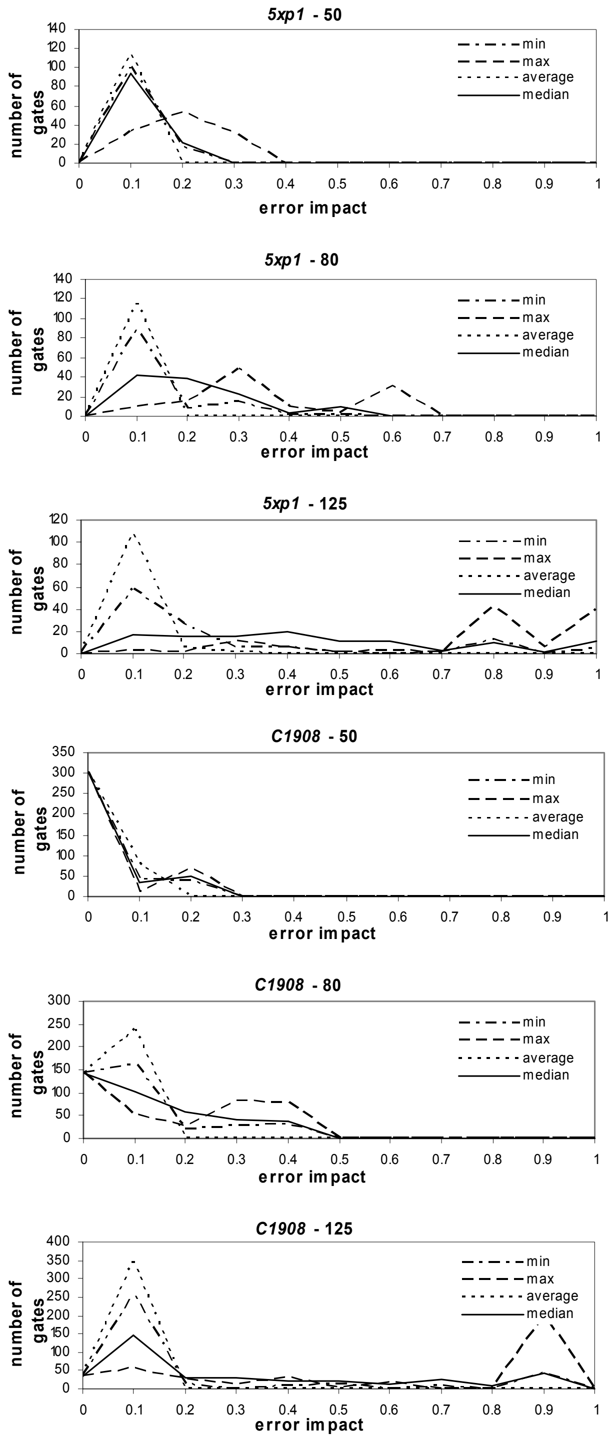

Figure 8. Mean error impact for a small benchmark (*5xp1* – top three charts) and a large benchmark (*C1908* – bottom three charts) computed as in equation (18) for three glitch durations.

References

[1] K. Mohanram and N. A. Touba. *Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits*. In Proc. of International Test Conference (ITC), pp. 893-901, 2003.

[2] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. *Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults*, In IEEE Transactions on Computers, Vol.49, No.3, pp. 230-245, March 2000.

[3] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. *Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic*. In Proc. of International Conference on Dependable Systems and Networks, pp. 389-398, 2002.

[4] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson. *On Latching Probability of Particle Induced Transients in Combinational Networks*. In Proc. of Fault-Tolerant Computing Symposium, pp. 340-349, 1994.

[5] J. F. Ziegler *et al. IBM experiments in Soft Fails in Computer Electronics (1978-1994)*. In IBM Journal of Research and Development, Vol 40, pp. 3-18, 1996.

[6] M. P. Baze and S. P. Buchner. *Attenuation of Single Event Induced Pulses in CMOS Combinational Logic*. In IEEE Transaction on Nuclear Science, Vol. 44, No. 6, pp. 2217-2223, December 1997.

[7] M. Omana, G. Papasso, D. Rossi, and C. Metra. *A Model for Transient Fault Propagation in Combinatorial Logic*. In Proc. of the 9th IEEE International On-Line Testing Symposium, IOLTS'03, pp. 111-115, July 2003.

[8] C. Zhao, X. Bai, and S. Dey. *A Scalable Soft Spot Analysis Methodology for Noise Effects in Nano-meter Circuits*. In Proc. of ACM/IEEE Design Automation Conference (DAC), pp. 894-899, June 2004.

[9] M. Zhang and N. R. Shanbhag. *A Soft Error rate Analysis (SERA) Methodology*. In Proc. of ACM/IEEE International Conference on Computer Aided Design (ICCAD), pp. 111-118, 2004.

[10] Y. S. Dhillon, A. U. Diril, and A. Chatterjee. *Soft-Error Tolerance Analysis and Optimization of Nanometer Circuits*. In Proc. of Design, Automation and Test in Europe (DATE), pp. 288-293, March 2005.

[11] S. Krishnaswamy, G. F. Viamonte, I. L. Markov, and J. P. Hayes. *Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices*. In Proc. of Design, Automation and Test in Europe (DATE), pp. 282-287, March 2005.

[12] F. Somenzi. *CUDD: The CU decision diagram package*. http://vlsi.colorado.edu/fabio/CUDD.

[13] R. E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. In IEEE Transaction on Computers, C-35-8, pp. 677-691, August 1986.

[14] K. S. Brace, R. L. Rudell, and R. E. Bryant. *Efficient Implementation of a BDD Package*. In Proc. of 27th ACM/IEEE Design Automation Conference (DAC), pp. 40-45, 1990.

[15] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang. *Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping*. In Proc of ACM/IEEE Design Automation Conference (DAC), pp. 54-60, June 1993.

[16] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi. *Algebraic Decision Diagrams and Their Applications*. In Proc. of ACM/IEEE International Conference on Computer Aided Design (ICCAD), pp. 188-191, November 1993.

[17] K. Nepal, H. Song, and R. I. Bahar. *RESTA: A Robust and Extendable Symbolic Timing Analysis Tool*. In Proc. of Great Lakes Symposium on VLSI (GLSVLSI), April 2004.

[18] J. M. Rabaey, A. Chandrakasan, B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2003.

[19] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. *New paradigm of predictive MOSFET and interconnect modeling for early circuit design*. In Proc. of IEEE Custom Integrated Circuits Conference (CICC), pp. 201-204, June 2000.

[20] *Berkeley Predictive Technology Model (BPTM)*: http://www-device.eecs.berkeley.edu/~ptm.

[21] I. Sutherland, B. Sproull and D. Harris. *Logical Effort: Designing FastCMOS Circuits*. Morgan Kaufmann Publishers, Inc., pp.5-15, 63-73, 1999.